

RAF7-8

增加一级索引文件的设计

为了增加存储容量，可以考虑使用多级索引文件的设计。具体思路如下：

1. 一级索引文件：记录每个数据文件的索引文件位置。
2. 二级索引文件：记录具体的记录位置，包括文件索引和文件内偏移量。

一级索引文件和二级索引文件的结构

1. 一级索引文件：记录二级索引文件的位置，每条记录包含二级索引文件的偏移量。
2. 二级索引文件：记录具体的记录位置，每条记录包含数据文件的索引和文件内的偏移量。

设计结构

- 一级索引文件：存储二级索引文件的偏移量。
- 二级索引文件：存储数据文件的索引和数据文件内的偏移量。

一级索引文件的结构

- 二级索引文件的偏移量（8 字节）

二级索引文件的结构

- 数据文件的索引（8 字节）
- 数据文件内的偏移量（8 字节）

示例代码

一级索引文件的处理

```
cpp
复制代码
class RecordWriter {
public:
    RecordWriter(const std::string &filepath) : base_filepath(filepath),
current_file_index(0), current_file_size(0) {
        openNewFile();
        openIndexFiles();
    }

    ~RecordWriter() {
        close();
    }

    size_t append(const std::string &value) {
```

```

        std::lock_guard<std::mutex> lock(file_mutex);

        uint32_t checksum = calculateChecksum(value);
        size_t length = value.size();
        size_t index = current_file_index;

        if (current_file_size + length + sizeof(length) + sizeof(checksum) >
MAX_FILE_SIZE) {
            openNewFile();
            index = current_file_index;
        }

        size_t offset = current_file_size;

        // 写入数据长度、数据和校验和
        current_file.write(reinterpret_cast<const char *>(&length), sizeof(length));
        current_file.write(value.data(), length);
        current_file.write(reinterpret_cast<const char *>(&checksum), sizeof(checksum));

        if (!current_file) {
            throw std::runtime_error("Failed to write record to file.");
        }

        current_file_size += sizeof(length) + length + sizeof(checksum);

        // 记录二级索引
        size_t secondary_index_offset = secondary_index_file.tellp();
        secondary_index_file.write(reinterpret_cast<const char *>(&index),
sizeof(index));
        secondary_index_file.write(reinterpret_cast<const char *>(&offset),
sizeof(offset));

        // 记录一级索引
        primary_index_file.write(reinterpret_cast<const char *>(&secondary_index_offset),
sizeof(secondary_index_offset));

        return index;
    }

    void close() {
        std::lock_guard<std::mutex> lock(file_mutex);
        if (current_file.is_open()) {
            current_file.close();
        }
        if (secondary_index_file.is_open()) {
            secondary_index_file.close();
        }
        if (primary_index_file.is_open()) {
            primary_index_file.close();
        }
    }
}

```

```

private:
    std::ofstream current_file;
    std::ofstream secondary_index_file;
    std::ofstream primary_index_file;
    std::mutex file_mutex;
    size_t current_file_index;
    size_t current_file_size;
    std::string base_filepath;

    void openNewFile() {
        if (current_file.is_open()) {
            current_file.close();
        }

        current_file_index++;
        current_file_size = 0;
        std::string file_name = base_filepath + "_" + std::to_string(current_file_index)
+ ".bin";
        current_file.open(file_name, std::ios::binary | std::ios::out);
        if (!current_file) {
            throw std::runtime_error("Failed to open new file for writing.");
        }
    }

    void openIndexFiles() {
        if (!secondary_index_file.is_open()) {
            secondary_index_file.open(base_filepath + "_secondary.idx", std::ios::binary
| std::ios::out | std::ios::app);
            if (!secondary_index_file) {
                throw std::runtime_error("Failed to open secondary index file for
writing.");
            }
        }

        if (!primary_index_file.is_open()) {
            primary_index_file.open(base_filepath + "_primary.idx", std::ios::binary |
std::ios::out | std::ios::app);
            if (!primary_index_file) {
                throw std::runtime_error("Failed to open primary index file for
writing.");
            }
        }
    }

    uint32_t calculateChecksum(const std::string &value) {
        uint32_t checksum = 0;
        for (char c : value) {
            checksum += static_cast<uint32_t>(c);
        }
        return checksum;
    }
};

```

二级索引文件的处理

cpp

复制代码

```
class RecordReader {
public:
    RecordReader(const std::string &filepath) : base_filepath(filepath) {
        loadPrimaryIndex();
        loadSecondaryIndex();
    }

    std::string read(size_t index) {
        std::lock_guard<std::mutex> lock(file_mutex);

        if (index >= primary_index_records.size()) {
            throw std::out_of_range("Index out of range.");
        }

        size_t secondary_index_offset = primary_index_records[index];
        size_t file_index, offset;

        secondary_index_file.seekg(secondary_index_offset, std::ios::beg);
        secondary_index_file.read(reinterpret_cast<char *>(&file_index),
sizeof(file_index));
        secondary_index_file.read(reinterpret_cast<char *>(&offset), sizeof(offset));

        std::string file_name = base_filepath + "_" + std::to_string(file_index) +
".bin";
        std::ifstream file(file_name, std::ios::binary | std::ios::in);
        if (!file) {
            throw std::runtime_error("Failed to open file for reading.");
        }

        file.seekg(offset, std::ios::beg);

        size_t length;
        file.read(reinterpret_cast<char *>(&length), sizeof(length));
        std::vector<char> value(length);
        file.read(value.data(), length);
        uint32_t checksum;
        file.read(reinterpret_cast<char *>(&checksum), sizeof(checksum));

        if (!file) {
            throw std::runtime_error("Failed to read record from file.");
        }

        // 校验和验证
        if (calculateChecksum(std::string(value.begin(), value.end())) != checksum) {
            throw std::runtime_error("Checksum verification failed.");
        }
    }
};
```

```

        return std::string(value.begin(), value.end());
    }

private:
    std::mutex file_mutex;
    std::vector<size_t> primary_index_records;
    std::string base_filepath;
    std::ifstream secondary_index_file;
    std::ifstream primary_index_file;

    void loadPrimaryIndex() {
        primary_index_file.open(base_filepath + "_primary.idx", std::ios::binary |
std::ios::in);
        if (!primary_index_file) {
            throw std::runtime_error("Failed to open primary index file for reading.");
        }

        size_t offset;
        while (primary_index_file.read(reinterpret_cast<char *>(&offset),
sizeof(offset))) {
            primary_index_records.push_back(offset);
        }
    }

    void loadSecondaryIndex() {
        secondary_index_file.open(base_filepath + "_secondary.idx", std::ios::binary |
std::ios::in);
        if (!secondary_index_file) {
            throw std::runtime_error("Failed to open secondary index file for reading.");
        }
    }

    uint32_t calculateChecksum(const std::string &value) {
        uint32_t checksum = 0;
        for (char c : value) {
            checksum += static_cast<uint32_t>(c);
        }
        return checksum;
    }
};

```

存储容量估算

1. 假设每条记录的数据长度为 **8 字节**，包括文件序号和文件内偏移量，每个记录 16 字节。
2. 索引文件：假设索引文件最大 4GB。
3. 一级索引文件：
 - 每条记录 8 字节。
 - $4\text{GB} / 8 \text{ 字节} = 536,870,912$ 条记录。

4. 二级索引文件：

- 每条记录 16 字节。
- $4\text{GB} / 16 \text{ 字节} = 268,435,456$ 条记录。

5. 总记录数：

- 可以存储 $536,870,912 * 268,435,456 = 144,115,188,075,855,872$ 条记录。

总结

通过增加一级索引文件，显著提升了系统的存储容量。每次数据记录存储在不同的数据文件中，索引文件记录了每条记录的位置，用户可以通过返回的索引准确地读取数据。该设计确保了系统可以处理更大的数据量，并且在性能和容量之间取得平衡。

4o

请你详细说一下一级索引文件个二级索引文件 以及最终的数据文件存的每条记录是什么 举一个写入和读取的例子 不用写代码 比如距离一条record 怎么写入的 以及写入后 这三种文件分别存的是什么东西 然后返回给用户的index是什么 然后举例子用户拿这个index是怎么一级一级读取到数据的

ChatGPT

一级索引文件、二级索引文件和数据文件

文件结构说明

1. 一级索引文件：

- 存储每个二级索引文件的偏移量。
- 每条记录大小：8 字节。
- 结构：[二级索引文件偏移量]

2. 二级索引文件：

- 存储具体的记录位置，包括数据文件的索引和数据文件内的偏移量。
- 每条记录大小：16 字节。
- 结构：[数据文件索引][文件内偏移量]

3. 数据文件：

- 存储实际的数据记录。
- 每条记录大小：数据长度 + 校验和长度（4 字节）+ 数据长度字段（4 字节）。
- 结构：[数据长度][数据][校验和]

写入和读取过程示例

写入过程

假设我们要写入一条记录 "Example record data"。

1. 计算校验和：

- 数据: "Example record data"

- 校验和 (假设为 12345)

2. 写入数据文件:

- 数据文件索引: 1

- 文件内偏移量: 0

- 写入内容: [数据长度(20)][数据("Example record data")][校验和(12345)]

3. 更新二级索引文件:

- 二级索引文件的偏移量: 0 (这是二级索引文件的第一个记录)

- 记录内容: [数据文件索引(1)][文件内偏移量(0)]

4. 更新一级索引文件:

- 一级索引文件的偏移量: 0 (这是一级索引文件的第一个记录)

- 记录内容: [二级索引文件偏移量(0)]

索引和数据文件的内容示例

- 一级索引文件 (假设偏移量为字节) :

```
arduino
复制代码
0x00: 0x0000000000000000 // 二级索引文件的偏移量
```

- 二级索引文件 (假设偏移量为字节) :

```
arduino
复制代码
0x00: 0x0000000000000001 // 数据文件索引
0x08: 0x0000000000000000 // 文件内偏移量
```

- 数据文件 (假设偏移量为字节) :

```
arduino
复制代码
0x00: 0x00000014          // 数据长度 (20 字节)
0x04: "Example record data" // 数据
0x18: 0x00003039          // 校验和 (12345)
```

返回给用户的索引

- 返回给用户的索引: 一级索引文件的记录号, 即 0。

读取过程

假设用户提供的索引为 0。

1. 读取一级索引文件:

- 从一级索引文件读取偏移量: 0x0000000000000000

2. 读取二级索引文件:

- 根据偏移量从二级索引文件读取记录：
 - 数据文件索引: `0x0000000000000001` (1)
 - 文件内偏移量: `0x0000000000000000` (0)

3. 读取数据文件:

- 打开数据文件 `datafile_1.bin`
- 从偏移量

```
0
```

读取数据:

- 数据长度: `0x00000014` (20 字节)
- 数据: `"Example record data"`
- 校验和: `0x00003039` (12345)

4. 校验和验证:

- 计算读取到的数据的校验和, 验证是否匹配。

具体示例

1. 写入记录:

- 写入数据 `"Example record data"`。
- 数据文件

```
datafile_1.bin
```

中写入:

```
arduino
复制代码
0x00: 0x00000014          // 数据长度 (20 字节)
0x04: "Example record data" // 数据
0x18: 0x00003039          // 校验和 (12345)
```

- 二级索引文件

```
secondary.idx
```

中写入:

```
arduino
复制代码
0x00: 0x0000000000000001 // 数据文件索引
0x08: 0x0000000000000000 // 文件内偏移量
```

- 一级索引文件


```
primary.idx
```

中写入：

```
arduino
复制代码
0x00: 0x0000000000000000 // 二级索引文件偏移量
```

2. 用户读取：

- 用户提供索引 0。
- 从一级索引文件 `primary.idx` 读取偏移量 0。
- 从二级索引文件

```
secondary.idx
```

读取记录：

```
arduino
复制代码
0x00: 0x0000000000000001 // 数据文件索引
0x08: 0x0000000000000000 // 文件内偏移量
```

- 从数据文件

```
datafile_1.bin
```

读取数据：

```
arduino
复制代码
0x00: 0x00000014          // 数据长度 (20 字节)
0x04: "Example record data" // 数据
0x18: 0x00003039          // 校验和 (12345)
```

通过这种设计，我们可以有效地管理大量数据记录，同时通过多级索引文件保持高效的查询性能。

40

二级索引文件也会写满吧 一级索引文件不需要记录二级索引文件号吗 应该记录二级索引文件号+偏移量吧

多级索引文件的设计

为了管理大量的数据记录，并且防止单个二级索引文件写满，可以采用多级索引文件设计，记录二级索引文件号和偏移量。

文件结构说明

1. 一级索引文件：

- 存储二级索引文件的位置和偏移量。
- 每条记录大小：16 字节。
- 结构：[二级索引文件号][二级索引文件偏移量]

2. 二级索引文件：

- 存储具体的记录位置，包括数据文件的索引和数据文件内的偏移量。
- 每条记录大小：16 字节。
- 结构：[数据文件索引][文件内偏移量]

3. 数据文件：

- 存储实际的数据记录。
- 每条记录大小：数据长度 + 校验和长度（4 字节） + 数据长度字段（4 字节）。
- 结构：[数据长度][数据][校验和]

写入和读取过程示例

写入过程

假设我们要写入一条记录 "Example record data"。

1. 计算校验和：

- 数据："Example record data"
- 校验和（假设为 12345）

2. 写入数据文件：

- 数据文件索引：1
- 文件内偏移量：0
- 写入内容：[数据长度(20)][数据("Example record data")][校验和(12345)]

3. 更新二级索引文件：

- 二级索引文件号：1
- 二级索引文件的偏移量：0（这是二级索引文件的第一个记录）
- 记录内容：[数据文件索引(1)][文件内偏移量(0)]

4. 更新一级索引文件：

- 一级索引文件的偏移量：0（这是一级索引文件的第一个记录）
- 记录内容：[二级索引文件号(1)][二级索引文件偏移量(0)]

索引和数据文件的内容示例

- 一级索引文件（假设偏移量为字节）：

```
arduino
复制代码
0x00: 0x0000000000000001 // 二级索引文件号
0x08: 0x0000000000000000 // 二级索引文件的偏移量
```

- 二级索引文件（假设偏移量为字节）：

```
arduino
复制代码
0x00: 0x0000000000000001 // 数据文件索引
0x08: 0x0000000000000000 // 文件内偏移量
```

- 数据文件（假设偏移量为字节）：

```
arduino
复制代码
0x00: 0x00000014           // 数据长度（20 字节）
0x04: "Example record data" // 数据
0x18: 0x00003039           // 校验和（12345）
```

返回给用户的索引

- 返回给用户的索引：一级索引文件的记录号，即 0。

读取过程

假设用户提供的索引为 0。

1. 读取一级索引文件：

- 从一级索引文件读取记录：
 - 二级索引文件号：0x0000000000000001 (1)
 - 二级索引文件偏移量：0x0000000000000000 (0)

2. 读取二级索引文件：

- 打开二级索引文件 secondary_1.idx
- 根据偏移量从二级索引文件读取记录：
 - 数据文件索引：0x0000000000000001 (1)
 - 文件内偏移量：0x0000000000000000 (0)

3. 读取数据文件：

- 打开数据文件 datafile_1.bin
- 从偏移量

0

读取数据：

- 数据长度：0x00000014 (20 字节)

- 数据: "Example record data"
- 校验和: 0x00003039 (12345)

4. 校验和验证:

- 计算读取到的数据的校验和, 验证是否匹配。

具体示例

1. 写入记录:

- 写入数据 "Example record data"。
- 数据文件

```
datafile_1.bin
```

中写入:

```
arduino
复制代码
0x00: 0x00000014          // 数据长度 (20 字节)
0x04: "Example record data" // 数据
0x18: 0x00003039          // 校验和 (12345)
```

- 二级索引文件

```
secondary_1.idx
```

中写入:

```
arduino
复制代码
0x00: 0x0000000000000001 // 数据文件索引
0x08: 0x0000000000000000 // 文件内偏移量
```

- 一级索引文件

```
primary.idx
```

中写入:

```
arduino
复制代码
0x00: 0x0000000000000001 // 二级索引文件号
0x08: 0x0000000000000000 // 二级索引文件偏移量
```

2. 用户读取:

- 用户提供索引 0。

- 从一级索引文件

```
primary.idx
```

读取记录：

```
arduino
复制代码
0x00: 0x0000000000000001 // 二级索引文件号
0x08: 0x0000000000000000 // 二级索引文件的偏移量
```

- 打开二级索引文件

```
secondary_1.idx
```

，从偏移量

```
0
```

读取记录：

```
arduino
复制代码
0x00: 0x0000000000000001 // 数据文件索引
0x08: 0x0000000000000000 // 文件内偏移量
```

- 打开数据文件

```
datafile_1.bin
```

，从偏移量

```
0
```

读取数据：

```
arduino
复制代码
0x00: 0x00000014           // 数据长度 (20 字节)
0x04: "Example record data" // 数据
0x18: 0x00003039           // 校验和 (12345)
```

通过这种多级索引文件设计，可以大幅度增加系统的存储容量，并确保数据的高效读写和管理。

再写入第二条数据（你自己随便举例子） 然后解释一下这三个文件存的是什么

ChatGPT

增加第二条记录后的文件内容示例

假设我们再写入一条记录 "Second record data"，如下详细描述写入和读取过程以及各个文件存储的内容。

写入过程

1. 第一条记录：

- 数据： "Example record data"
- 校验和： 12345
- 数据文件： datafile_1.bin
- 偏移量： 0

2. 第二条记录：

- 数据： "Second record data"
- 校验和： 67890
- 数据文件： datafile_1.bin
- 偏移量： 0x1C (28 字节)

数据文件 datafile_1.bin

1. 第一条记录：

```
arduino
复制代码
0x00: 0x00000014 // 数据长度 (20 字节)
0x04: "Example record data" // 数据
0x18: 0x00003039 // 校验和 (12345)
```

2. 第二条记录：

```
arduino
复制代码
0x1C: 0x00000013 // 数据长度 (19 字节)
0x20: "Second record data" // 数据
0x33: 0x00010932 // 校验和 (67890)
```

二级索引文件 secondary_1.idx

1. 第一条记录：

```
arduino
复制代码
0x00: 0x0000000000000001 // 数据文件索引 (1)
0x08: 0x0000000000000000 // 文件内偏移量 (0)
```

2. 第二条记录:

```
arduino
复制代码
0x10: 0x0000000000000001 // 数据文件索引 (1)
0x18: 0x000000000000001C // 文件内偏移量 (28 字节)
```

一级索引文件 `primary.idx`

1. 第一条记录:

```
arduino
复制代码
0x00: 0x0000000000000001 // 二级索引文件号 (1)
0x08: 0x0000000000000000 // 二级索引文件偏移量 (0)
```

2. 第二条记录:

```
arduino
复制代码
0x10: 0x0000000000000001 // 二级索引文件号 (1)
0x18: 0x0000000000000010 // 二级索引文件偏移量 (16 字节)
```

读取过程

假设用户提供索引 `1` (第二条记录)。

1. 读取一级索引文件:

- 索引: `1`
- 二级索引文件号: `0x0000000000000001` (1)
- 二级索引文件偏移量: `0x0000000000000010` (16 字节)

2. 读取二级索引文件:

- 二级索引文件号: `1`
- 二级索引文件偏移量: `16`
- 数据文件索引: `0x0000000000000001` (1)
- 文件内偏移量: `0x000000000000001C` (28 字节)

3. 读取数据文件:

- 数据文件: `datafile_1.bin`
- 偏移量: `28`
- 数据长度: `0x00000013` (19 字节)
- 数据: `"Second record data"`

- 校验和: `0x00010932` (67890)

详细解释

- **数据文件:** `datafile_1.bin` 存储所有记录的数据。每条记录包含数据长度、数据内容和校验和。
- **二级索引文件:** `secondary_1.idx` 存储数据文件中每条记录的索引信息, 包括数据文件的索引和文件内的偏移量。
- **一级索引文件:** `primary.idx` 存储指向二级索引文件的记录位置, 包括二级索引文件号和二级索引文件内的偏移量。

用户返回的索引

- 返回给用户的索引是一级索引文件中的记录号。
- 用户通过一级索引文件找到对应的二级索引文件号和偏移量。
- 然后通过二级索引文件找到数据文件的索引和偏移量。
- 最后通过数据文件的偏移量读取具体的数据。

通过这种多级索引设计, 可以有效地管理大量数据记录, 并确保高效的读写操作。

40

那基于这样的二级索引 代码写出来 writer和reader类分别有appendBatch和readBatch函数 别忘了实现 给出详细的代码注释 最好每一段逻辑都补上注释 不要偷懒

ChatGPT

多级索引实现: Writer 和 Reader 类代码

下面是详细的 `RecordWriter` 和 `RecordReader` 类实现, 包括 `appendBatch` 和 `readBatch` 函数。每段代码都附有注释, 解释逻辑。

RecordWriter 类

```
cpp
复制代码
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <mutex>
#include <stdexcept>

const std::string FILE_PREFIX = "datafile_";
const std::string PRIMARY_INDEX_FILE = "primary.idx";
const std::string SECONDARY_INDEX_FILE_PREFIX = "secondary_";
const size_t MAX_FILE_SIZE = 1024 * 1024 * 1024; // 1GB
```



```

class RecordWriter {
public:
    RecordWriter(const std::string &filepath) : base_filepath(filepath),
current_file_index(0), current_file_size(0), secondary_file_index(0) {
        openNewFile();
        openPrimaryIndexFile();
    }

    ~RecordWriter() {
        close();
    }

    size_t append(const std::string &value) {
        std::lock_guard<std::mutex> lock(file_mutex);

        uint32_t checksum = calculateChecksum(value);
        size_t length = value.size();
        size_t data_file_index = current_file_index;

        if (current_file_size + length + sizeof(length) + sizeof(checksum) >
MAX_FILE_SIZE) {
            openNewFile();
            data_file_index = current_file_index;
        }

        size_t offset = current_file_size;

        // 写入数据长度、数据和校验和
        current_file.write(reinterpret_cast<const char *>(&length), sizeof(length));
        current_file.write(value.data(), length);
        current_file.write(reinterpret_cast<const char *>(&checksum), sizeof(checksum));

        if (!current_file) {
            throw std::runtime_error("Failed to write record to file.");
        }

        current_file_size += sizeof(length) + length + sizeof(checksum);

        // 更新二级索引文件
        std::ofstream secondary_index_file = openSecondaryIndexFile();
        size_t secondary_index_offset = secondary_index_file.tellp();
        secondary_index_file.write(reinterpret_cast<const char *>(&data_file_index),
sizeof(data_file_index));
        secondary_index_file.write(reinterpret_cast<const char *>(&offset),
sizeof(offset));
        secondary_index_file.close();

        // 更新一级索引文件
        primary_index_file.write(reinterpret_cast<const char *>(&secondary_file_index),
sizeof(secondary_file_index));
    }
};

```

```

        primary_index_file.write(reinterpret_cast<const char *>(&secondary_index_offset),
sizeof(secondary_index_offset));

        return current_file_index;
    }

    void appendBatch(const std::vector<std::string> &values) {
        for (const auto &value : values) {
            append(value);
        }
    }

    void close() {
        std::lock_guard<std::mutex> lock(file_mutex);
        if (current_file.is_open()) {
            current_file.close();
        }
        if (primary_index_file.is_open()) {
            primary_index_file.close();
        }
    }

private:
    std::ofstream current_file;
    std::ofstream primary_index_file;
    std::mutex file_mutex;
    size_t current_file_index;
    size_t current_file_size;
    size_t secondary_file_index;
    std::string base_filepath;

    void openNewFile() {
        if (current_file.is_open()) {
            current_file.close();
        }

        current_file_index++;
        current_file_size = 0;
        std::string file_name = base_filepath + "_" + std::to_string(current_file_index)
+ ".bin";
        current_file.open(file_name, std::ios::binary | std::ios::out);
        if (!current_file) {
            throw std::runtime_error("Failed to open new file for writing.");
        }
    }

    void openPrimaryIndexFile() {
        if (!primary_index_file.is_open()) {
            primary_index_file.open(base_filepath + "_" + PRIMARY_INDEX_FILE,
std::ios::binary | std::ios::out | std::ios::app);
            if (!primary_index_file) {

```

```

        throw std::runtime_error("Failed to open primary index file for
writing.");
    }
}

std::ofstream openSecondaryIndexFile() {
    secondary_file_index = current_file_index;
    std::string file_name = base_filepath + "_" + SECONDARY_INDEX_FILE_PREFIX +
std::to_string(secondary_file_index) + ".idx";
    std::ofstream secondary_index_file(file_name, std::ios::binary | std::ios::out |
std::ios::app);
    if (!secondary_index_file) {
        throw std::runtime_error("Failed to open secondary index file for writing.");
    }
    return secondary_index_file;
}

uint32_t calculateChecksum(const std::string &value) {
    uint32_t checksum = 0;
    for (char c : value) {
        checksum += static_cast<uint32_t>(c);
    }
    return checksum;
}
};

```

RecordReader 类

```

cpp
复制代码
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <mutex>
#include <stdexcept>

class RecordReader {
public:
    RecordReader(const std::string &filepath) : base_filepath(filepath) {
        loadPrimaryIndex();
    }

    std::string read(size_t index) {
        std::lock_guard<std::mutex> lock(file_mutex);

        if (index >= primary_index_records.size()) {
            throw std::out_of_range("Index out of range.");
        }
    }
}

```

```

        size_t secondary_file_index = primary_index_records[index].first;
        size_t secondary_index_offset = primary_index_records[index].second;

        std::ifstream secondary_index_file =
openSecondaryIndexFile(secondary_file_index);
        size_t data_file_index, offset;
        secondary_index_file.seekg(secondary_index_offset, std::ios::beg);
        secondary_index_file.read(reinterpret_cast<char *>(&data_file_index),
sizeof(data_file_index));
        secondary_index_file.read(reinterpret_cast<char *>(&offset), sizeof(offset));
        secondary_index_file.close();

        return readDataFile(data_file_index, offset);
    }

    std::vector<std::string> readBatch(const std::vector<size_t> &indexes) {
        std::vector<std::string> results;
        for (const auto &index : indexes) {
            results.push_back(read(index));
        }
        return results;
    }

private:
    std::mutex file_mutex;
    std::vector<std::pair<size_t, size_t>> primary_index_records;
    std::string base_filepath;

    void loadPrimaryIndex() {
        std::ifstream primary_index_file(base_filepath + "_" + PRIMARY_INDEX_FILE,
std::ios::binary | std::ios::in);
        if (!primary_index_file) {
            throw std::runtime_error("Failed to open primary index file for reading.");
        }

        size_t secondary_file_index, secondary_index_offset;
        while (primary_index_file.read(reinterpret_cast<char *>(&secondary_file_index),
sizeof(secondary_file_index)) &&
            primary_index_file.read(reinterpret_cast<char *>(&secondary_index_offset),
sizeof(secondary_index_offset))) {
            primary_index_records.emplace_back(secondary_file_index,
secondary_index_offset);
        }
    }

    std::ifstream openSecondaryIndexFile(size_t index) {
        std::string file_name = base_filepath + "_" + SECONDARY_INDEX_FILE_PREFIX +
std::to_string(index) + ".idx";
        std::ifstream secondary_index_file(file_name, std::ios::binary | std::ios::in);
        if (!secondary_index_file) {
            throw std::runtime_error("Failed to open secondary index file for reading.");
        }
    }

```

```

        return secondary_index_file;
    }

    std::string readDataFile(size_t file_index, size_t offset) {
        std::string file_name = base_filepath + "_" + std::to_string(file_index) +
".bin";
        std::ifstream data_file(file_name, std::ios::binary | std::ios::in);
        if (!data_file) {
            throw std::runtime_error("Failed to open data file for reading.");
        }

        data_file.seekg(offset, std::ios::beg);

        size_t length;
        data_file.read(reinterpret_cast<char *>(&length), sizeof(length));
        std::vector<char> value(length);
        data_file.read(value.data(), length);
        uint32_t checksum;
        data_file.read(reinterpret_cast<char *>(&checksum), sizeof(checksum));

        if (!data_file) {
            throw std::runtime_error("Failed to read record from file.");
        }

        // 校验和验证
        if (calculateChecksum(std::string(value.begin(), value.end())) != checksum) {
            throw std::runtime_error("Checksum verification failed.");
        }

        return std::string(value.begin(), value.end());
    }

    uint32_t calculateChecksum(const std::string &value) {
        uint32_t checksum = 0;
        for (char c : value) {
            checksum += static_cast<uint32_t>(c);
        }
        return checksum;
    }
};

```

注释解释

RecordWriter 类

1. **append**: 将单个记录追加到数据文件中。如果当前文件大小超过最大限制，则打开新文件。更新二级索引文件和一级索引文件。
2. **appendBatch**: 批量追加记录，调用 `append` 函数。
3. **openNewFile**: 打开一个新的数据文件。
4. **openPrimaryIndexFile**: 打开或创建一级索引文件。
5. **openSecondaryIndexFile**: 打开或创建二级索引文件。

6. **calculateChecksum**: 计算记录的校验和。

RecordReader 类

1. **read**: 根据索引读取单个记录。读取一级索引文件以获取二级索引文件号和偏移量，然后读取二级
多级索引实现: Writer 和 Reader 类代码

解决 **fio** 不显示后续任务结果的问题

在查看你的 **fio** 配置文件后，发现了一些可能导致问题的设置。我们来分析并调整配置文件，确保所有测试任务都能顺利运行并显示结果。

当前 **fio** 配置文件

```
ini
复制代码
[global]
ioengine=libaio
direct=1
invalidate=1
bs=4k
size=1G
numjobs=4
runtime=60
group_reporting

[seq_test]
rw=read
bs=4k
size=1G
filename=fio_1_test_file

[rand_test]
rw=randread
bs=4k
size=1G
filename=fio_2_test_file
```

问题可能的原因

1. **任务名称冲突**: 虽然这不太可能导致任务未显示结果，但确保每个任务名称（如 **[seq_test]** 和 **[rand_test]**）唯一是个好习惯。
2. **文件名冲突**: **filename** 参数应该指向不同的文件，以避免冲突。
3. **I/O 引擎和深度设置**: 确保 I/O 引擎和深度设置正确。
4. **运行命令**: 确保 **fio** 命令正确执行并包含所有任务。

调整后的配置文件

我们可以稍微调整配置文件，确保每个任务独立运行并输出结果。

```
ini
复制代码
[global]
ioengine=libaio
direct=1
invalidate=1
bs=4k
size=1G
numjobs=4
runtime=60
group_reporting

[seq_read]
rw=read
bs=4k
size=1G
filename=fio_seq_read_test_file

[rand_read]
rw=randread
bs=4k
size=1G
filename=fio_rand_read_test_file

[seq_write]
rw=write
bs=4k
size=1G
filename=fio_seq_write_test_file
```

确保缓存已清空

在每次运行 **fio** 测试前，清空系统缓存：

```
sh
复制代码
sync; echo 3 | sudo tee /proc/sys/vm/drop_caches
```

运行 **fio** 测试

确保命令正确运行，并输出所有任务的结果：

```
sh
复制代码
fio test.fio
```

分析 **fio** 输出结果

如果仍然只显示第一个任务的结果，请确保没有语法错误，并且所有文件名和任务名称是唯一的。还可以尝试分别运行每个任务，确保每个任务单独运行时没有问题。

```
sh
```

```
复制代码
```

```
fio --name=seq_read --rw=read --bs=4k --size=1G --numjobs=4 --runtime=60 --  
ioengine=libaio --direct=1 --invalidate=1 --filename=fio_seq_read_test_file  
fio --name=rand_read --rw=randread --bs=4k --size=1G --numjobs=4 --runtime=60 --  
ioengine=libaio --direct=1 --invalidate=1 --filename=fio_rand_read_test_file  
fio --name=seq_write --rw=write --bs=4k --size=1G --numjobs=4 --runtime=60 --  
ioengine=libaio --direct=1 --invalidate=1 --filename=fio_seq_write_test_file
```

输出结果解析

如果每个任务单独运行正常，那么可能是 `fio` 配置文件中某些设置导致问题。确保配置文件没有冲突，并重新运行所有任务：

```
sh
```

```
复制代码
```

```
fio test.fio
```

通过上述步骤，可以确保 `fio` 运行多个任务并正确输出结果